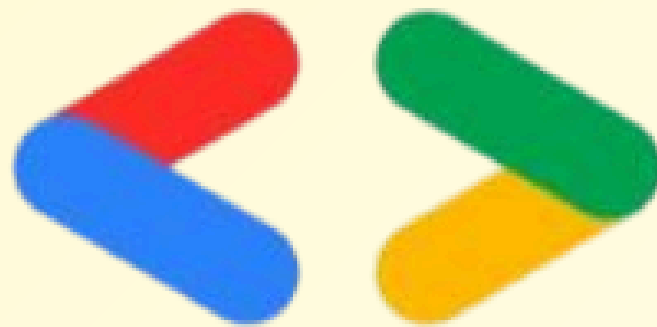


DEVSPRINT

Data Science - Machine Learning Workshop



Google Developer Groups

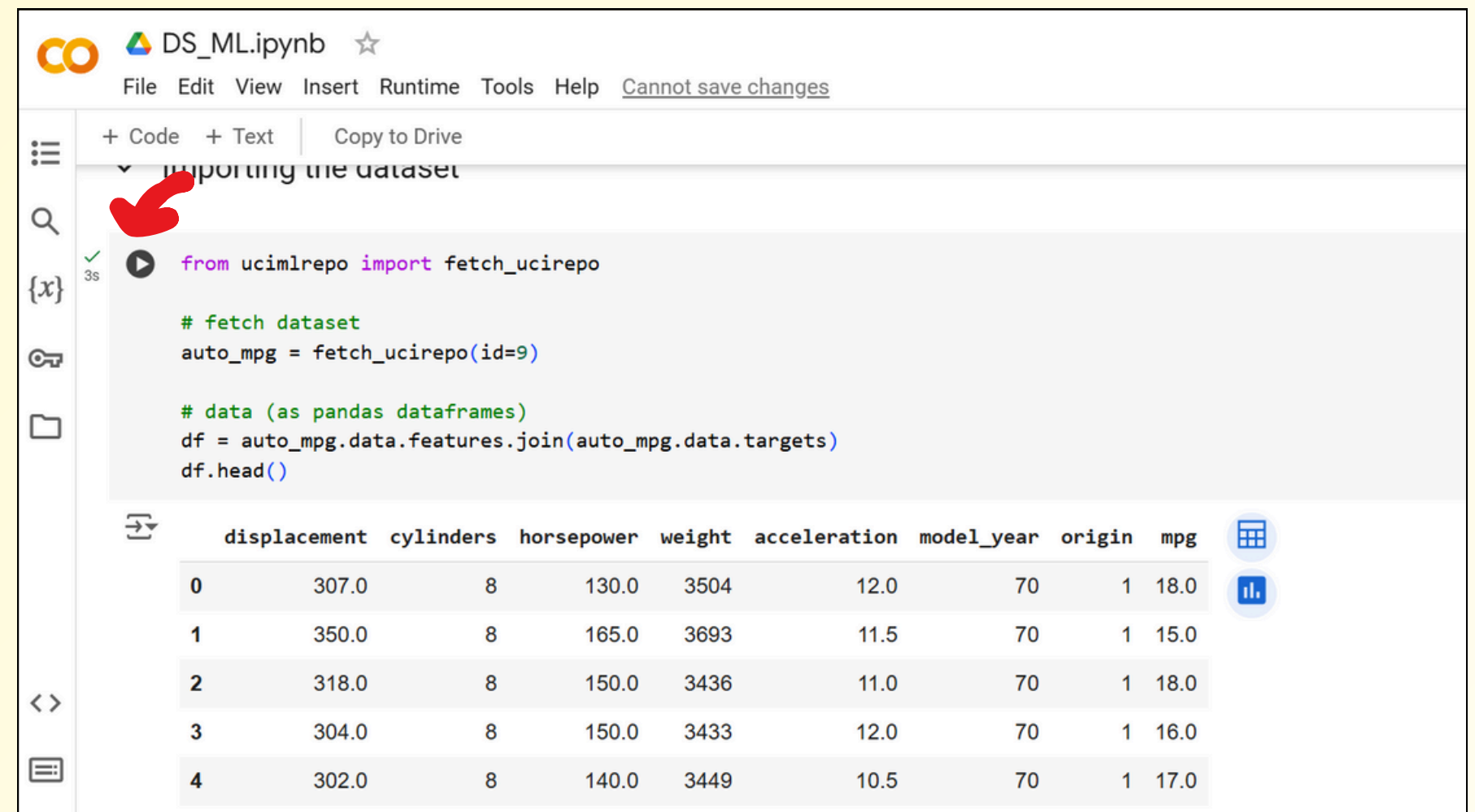
On Campus • National Institute Of Technology Goa

Introduction to Google Colab

Free, cloud-based platform provided by Google that allows you to write and execute Python code in a Jupyter Notebook-like interface

It is **cloud-based** and comes with **pre-installed libraries**, so we do not need to download any libraries or dependencies on our machine, unlike Jupyter Notebooks.

Just write the code in the **cells** provided, and then either click on that small **arrow** icon, or press **Shift+Enter** to execute the code.



```
from ucimlrepo import fetch_ucirepo

# fetch dataset
auto_mpg = fetch_ucirepo(id=9)

# data (as pandas dataframes)
df = auto_mpg.data.features.join(auto_mpg.data.targets)
df.head()
```

	displacement	cylinders	horsepower	weight	acceleration	model_year	origin	mpg
0	307.0	8	130.0	3504	12.0	70	1	18.0
1	350.0	8	165.0	3693	11.5	70	1	15.0
2	318.0	8	150.0	3436	11.0	70	1	18.0
3	304.0	8	150.0	3433	12.0	70	1	16.0
4	302.0	8	140.0	3449	10.5	70	1	17.0

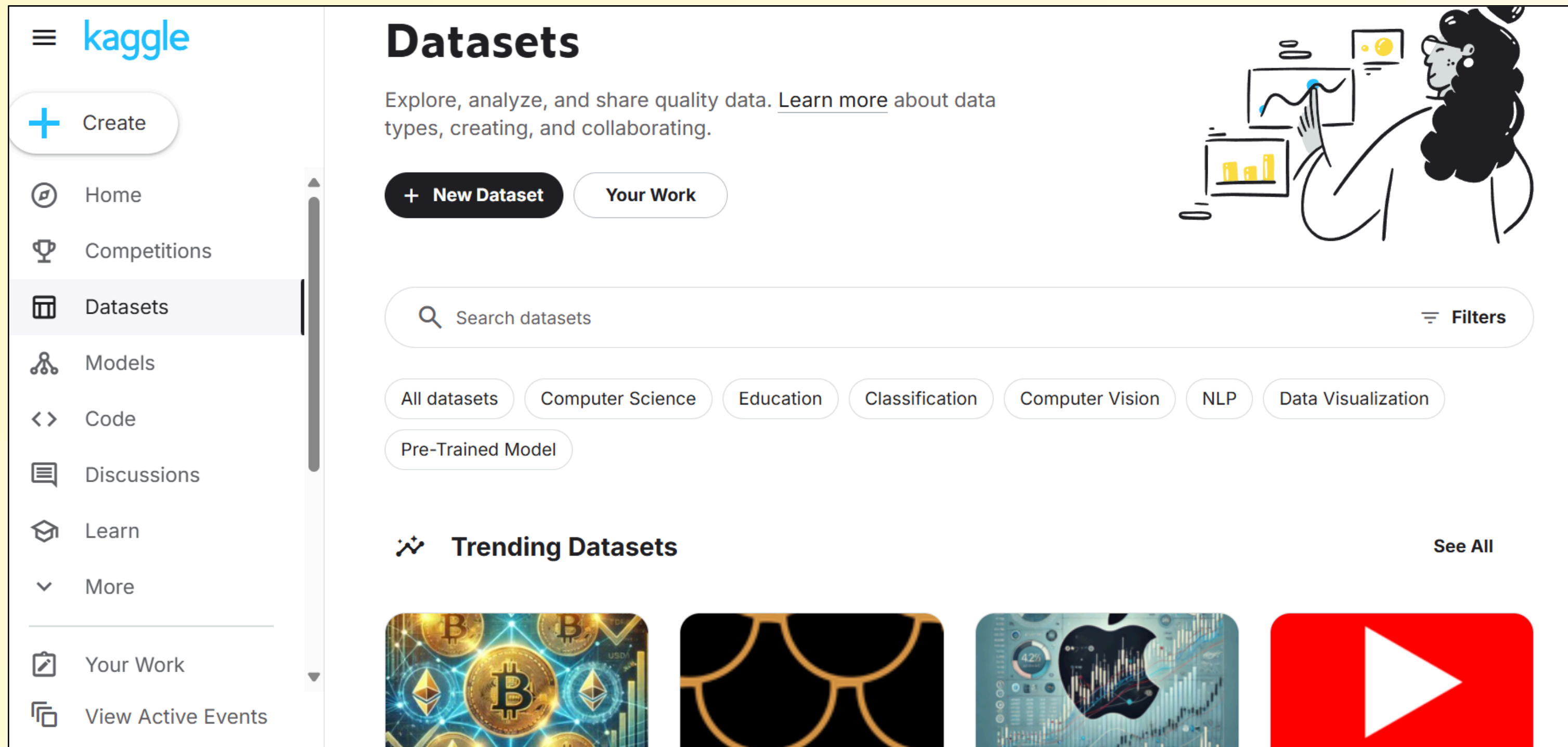
Agenda :

1. Choosing a Dataset.
2. Exploratory Data Analysis and Data Preprocessing using Python.
3. Applying ML algorithms linear regression and Random Forest for prediction.

Choosing Dataset


Popular Platforms for Datasets :

1. **Kaggle** : <https://www.kaggle.com/datasets>



2. UC Irvine Machine Learning Repository :

<https://archive.ics.uci.edu/>


DatasetsContribute DatasetAbout Us→ Login

Welcome to the UC Irvine Machine Learning Repository

We currently maintain 674 datasets as a service to the machine learning community. Here, you can donate and find datasets used by millions of people all around the world!

[VIEW DATASETS](#)[CONTRIBUTE A DATASET](#)


Popular Datasets



Iris

A small classic dataset from Fisher, 1936. One of the earliest known dat...

Classification150 Instances4 Features




Heart Disease

4 databases: Cleveland, Hungary, Switzerland, and the VA Long Beach

Classification303 Instances13 Features


New Datasets



Gas sensor array low-concentration

This dataset contains 6 gas responses collected by a sensor array consi...

Classification, Re...90 Instances



Twitter Geospatial Data

Seven days of geo-tagged Tweet data from the United States with exac...

Classification, Re...14.26M Instances4 Features

Chosen Dataset :

Dataset Name : AUTO mpg

Predicting the mpg
(fuel consumption in
miles per gallon) of
cars using machine
learning techniques.

	displacement	cylinders	horsepower	weight	acceleration	model_year	origin	mpg
0	307.0	8	130.0	3504	12.0	70	1	18.0
1	350.0	8	165.0	3693	11.5	70	1	15.0
2	318.0	8	150.0	3436	11.0	70	1	18.0
3	304.0	8	150.0	3433	12.0	70	1	16.0
4	302.0	8	140.0	3449	10.5	70	1	17.0
...
393	140.0	4	86.0	2790	15.6	82	1	27.0
394	97.0	4	52.0	2130	24.6	82	2	44.0
395	135.0	4	84.0	2295	11.6	82	1	32.0
396	120.0	4	79.0	2625	18.6	82	1	28.0
397	119.0	4	82.0	2720	19.4	82	1	31.0

398 rows × 8 columns

Data Preprocessing

1. Importing the relevant libraries

```
import pandas as pd  
import numpy as np
```

Pandas : Python library for processing data in the form of dataframes. Dataframes are like tables.

Numpy: Python library for processing data in the form of n-dimensional special arrays called numpy arrays.

2. Importing the Dataset.

We import the dataset from the UCI Machine Learning Repository in the form of a pandas DataFrame.

df.head(n)

Used to view the top n rows of the dataset. Default value of n is 5

3. Checking for NULL Values

Our dataset can have NULL or missing values which we must remove before the data goes as input to the model.

Hence, we first check for NULL values, and then remove them if there are any.

df.info() : Gives information about the dataset columns/attributes like data types, how many NULL values are there etc.

df.isnull() : Tells which values in the dataset are NULL and which are not. i.e. True for Null and False for Not Null

df.isnull.sum() : Sum of each column in df.isnull() i.e. Hence it tells how many NULL values are in each column.

df=df.dropna() : drops all the rows having NULL values from the dataset.

4. Exploratory Data Analysis

Here we check out for some statistics of the data, to help us understand the data better.

5. Splitting Data into inputs and outputs i.e. x and y.

df.iloc[rows-index , column-index]

- Index based data filtering. i.e. gets the data whose indices are in iloc.
- : means all
- For a Range of Values we write a:b to get data from indices [a,b)
- So **df.iloc[:, 0:7]** means getting the data from all rows, but columns from index 0 to index 6.

- Similarly, if we want all data from start to a particular index i , then we write $:i$ and if we want data from index i to end of dataframe, we write $i:$
- So `df.iloc[:, 7:]` means getting the data from all rows, but columns from index 7 onwards all columns.

6. Encoding Categorical Variables

- Categorical Variables are those, which are not numerical.
- We need to somehow convert them to numerical values so that we can feed them to the model.
- We use One-hot encoding for this.

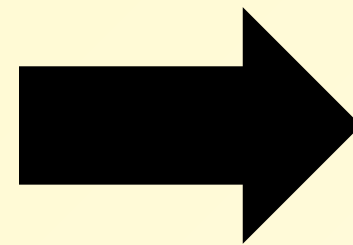
One-hot encoding

Let us consider an example.

Our dataset has a column called **Fruits** and can take only 3 values - Apple, Mango and Banana.

To One-hot encode these 3 values, we will **remove the fruits column**, and **add two more columns**- one for apple and one for mango.

FRUITS
Mango
Apple
Banana



Apple	Mango
0	1
1	0
0	0

We don't make a separate column for banana, because it is implied from the other 2 columns, and we don't need a separate column for it. If both apple and mango are 0, it means banana.

In our dataset, the column **Origin** has categorical values : 1,2 and 3. These maybe numbers, but don't hold any special meaning. They are like Categories. Category-1, Category-2 and Category-3. So we can't keep them as it is and need to encode them.

```
categorical = pd.get_dummies(df['origin'],drop_first=True)
```

- This function is used to **get dummies/encoded values** for the origin column.
- **drop_first = True** is used to drop one column so that we do not get one more column for the last variable(like banana).
- Then we drop the original column from the dataset, and add these new columns to it.

`x=x.drop(['origin'],axis=1)` : Drops that column from the dataset. axis =1 means drop that column axis=0 means drop that row.

`x=x.join(categorical)` : Joins the encoded columns that we got from `pd.get_dummies(..)` to the dataset.

7. Feature Scaling

- Feature Scaling is like telling the model to treat every column equally while making the predictions.
- Some columns may have extremely large values like **weight** can have values of magnitude 3509 and 2267 whereas some columns have extremely small values like **cylinders** has values such as 3 and 4.
- So the model, while making predictions will give more weightage to the weight column and less to the cylinders column, and this is **wrong!**

Hence we standardize the values in each column by subtracting the mean of the column from that value and dividing it by the standard deviation of that column.

$$z = \frac{x - \mu}{\sigma}$$

μ = Mean

σ = Standard Deviation

Now, every column has values only **between -1 and 1**, making every column equivalent(of the same scale).

First we import the StandardScaler class from the sklearn.preprocessing module

```
from sklearn.preprocessing import StandardScaler
```

Next we create an object of this class

```
scaler=StandardScaler()
```

scaler.fit(x) -> Computes the mean and std. deviation of every column

scaler.transform(x) -> Applies the formula to every value in x

scaler.fit_transform(x) -> Does both these together

```
x=scaler.fit_transform(x)
```


RECAP :

- We chose a dataset and imported it onto the notebook.
- Checked for, and removed NULL Values.
- Performed some EDA (exploratory data analysis) on the data.
- Split the data into x (inputs) and y (output).
- Encoded the categorical variables into one-hot encoded values.
- Applied feature scaling to the data.

Now, our data is ready to be input to the model.

Machine Learning

1. Importing the relevant libraries

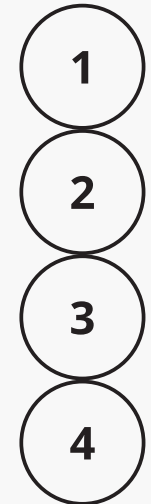
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

Scikit Learn : Scikit-learn is a machine learning library in Python that provides simple and efficient tools for data analysis and modeling, including algorithms for classification, regression, clustering, and more.

Matplotlib: Matplotlib is a plotting library that enables the creation of static, interactive, and animated visualizations in Python.

Splitting the dataset

```
X_train, X_test, y_train, y_test = train_test_split(X,  
                                                    y,  
                                                    test_size=0.2,  
                                                    random_state=42)
```



1. **X:** This is the feature set or the independent variable(s) of your dataset.
2. **y:** This is the target set or the dependent variable(s) you aim to predict.
3. **test_size:** Specifies the proportion of the dataset to include in the test split.
4. **random_state:** This controls the shuffling applied to the data before splitting.

Output:

X_train: 80% of X for training.

X_test: 20% of X for testing.

y_train: 80% of y for training.

y_test: 20% of y for testing.

Linear Regression

1

```
lr_model=LinearRegression()
```

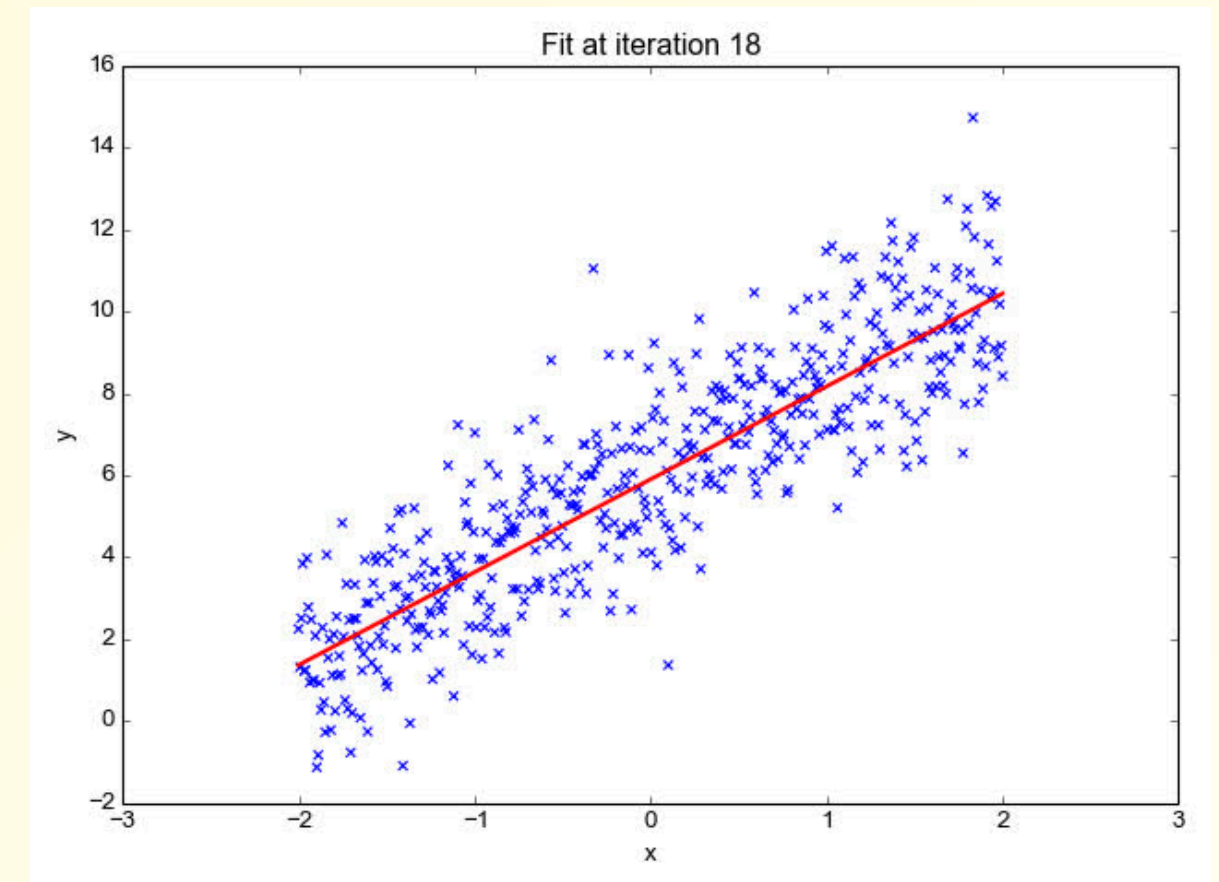
2

```
lr_model.fit(X_train, y_train)
```

3

```
lr_pred = lr_model.predict(X_test)
```

1. Creating a model
2. Fitting a model based on the training data
3. Putting the test data to see if predictions are correct



Getting the metrics of a model

Root Mean Square Error(RMSE)

```
lr_mse = mean_squared_error(y_test, lr_pred)
lr_rmse = np.sqrt(lr_mse)
```

$$RMSE = \sqrt{\sum_{i=1}^n \frac{(\hat{y}_i - y_i)^2}{n}}$$

Key Points:

- RMSE has the same unit as the target variable
- Closer to Zero, the better the model

R^2 Score

```
lr_r2 = r2_score(y_test, lr_pred)
```

$$R^2 = \frac{\text{RegSS}}{\text{TSS}} = \frac{\sum_i (\hat{y}_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2}$$

Key Points:

- R2=1: Perfect fit; the model explains all the variability in the data.
- R2=0: The model does no better than the mean of y
- R2<0: The model performs worse than the baseline mean model.

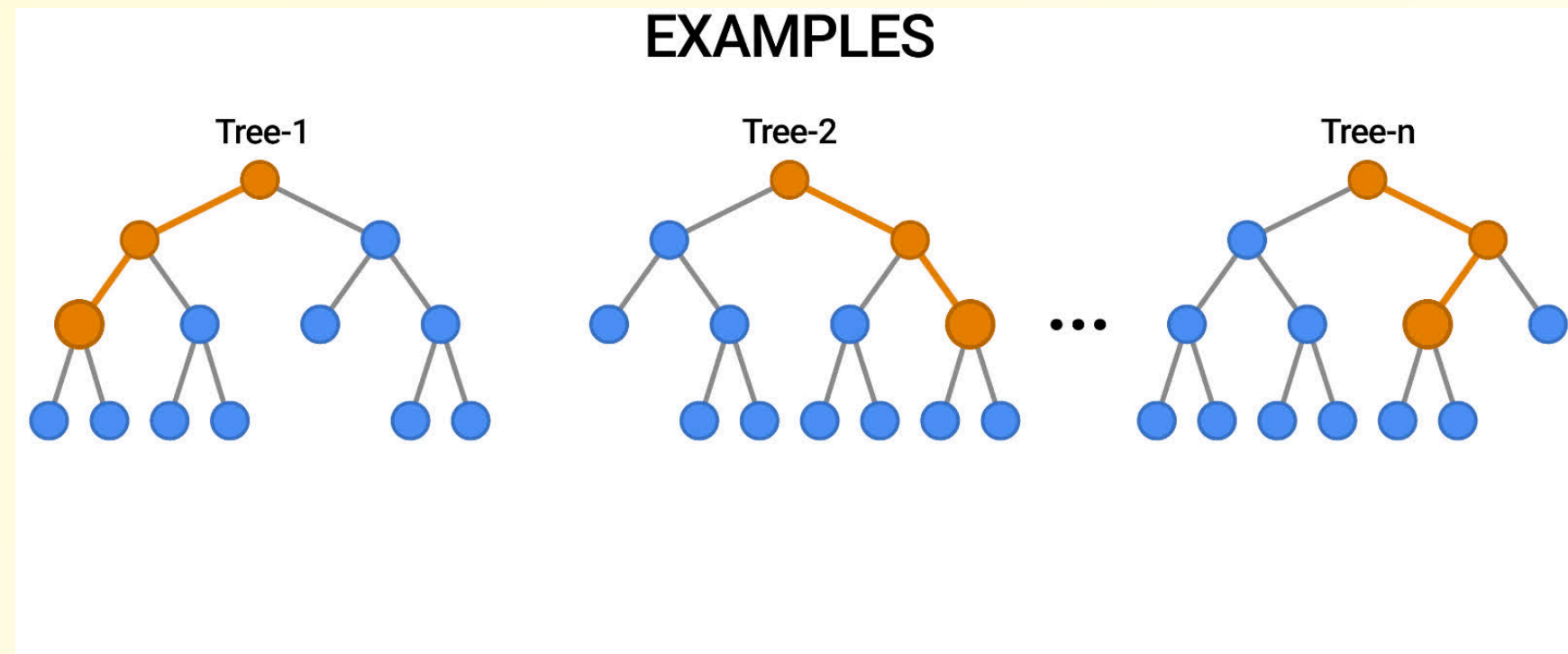
Plotting the Model Predictions



Random Forests

- 1 `rf_model = RandomForestRegressor(n_estimators=100, random_state=42)`
- 2 `rf_model.fit(X_train, y_train)`
- 3 `rf_pred = rf_model.predict(X_test)`

1. Creating a model
2. Fitting a model based on the training data
3. Putting the test data to see if predictions are correct



Cross Validation

```
from sklearn.model_selection import cross_val_score  
  
cv_scores = cross_val_score(rf_model, X, y, cv=5, scoring='r2')  
  
print(f"Average R2: {cv_scores.mean():.2f}")
```

Cross-validation is a technique used to evaluate a machine learning model's performance and ability to generalize by splitting the dataset into multiple subsets (folds). The model is trained and tested multiple times, each time using different training and testing data, ensuring it performs well on unseen data.

What next?

- Read the documentation
- Try applying more data processing techniques as well as machine learning techniques to different data sets
- Experiment with different ML models